Unity REST API

The Unity REST API enables programmatic access to Unisphere management capabilities.

Version: 5.2.0 **Category: Storage**

TUTORIALS

Introduction to Unity REST API

REST stands for "Representational State Transfer". It is a client-server architectural style for designing services over HTTP in a simple, effective way.

The Unity REST API provides a set of resources, operations, and attributes so you could easy to interact with Unity.

Make your first call

The Unity REST API is powerful but very easy to use, we could use curl command to make a very basic query request to get all pools in Unity in a second:

bash
```bash
curl -i -k -L \
  -u admin:Password123! \
  -c cookie.txt \
  -H "Accept: application/json" \
  -H "Content-Type: application/json" \
  -H "X-EMC-REST-CLIENT: true" \
  "https://10.245.83.44:443/api/types/pool/instances?compact=True&fields=id,name"
```

Unity will return all of the pool ids and names:

json
```json
{
  "@base": "https://10.245.83.44/api/types/pool/instances?
fields=id,name&per_page=2000&compact=true",
  "updated": "2020-08-27T00:25:45.443Z",
  "links": [
    {
      "rel": "self",
      "href": "&page=1"
    }
  ],
  "entries": [
    {
      "content": {
        "id": "pool_1",
        "name": "Cinder_Pool"
      }
    },
    {
      "content": {
        "id": "pool_2",
        "name": "Manila_Pool"
      }
    },
    {
      "content": {
        "id": "pool_3",
        "name": "Flash_Pool"
      }
    }
  ]
}
```

Let's understand the curl parameters and URL one by one:

- `-i`: include protocol response headers in the output, optional
- `-k`: allow insecure server connections when using SSL
- `-L`: handle the HTTP redirections, required
- `-u admin:Password123!`: specify the server user and password, required
- `-c cookie.txt`: write cookie to a file for further usage, required
- `-H &quot;Accept: application/json&quot;`: indicate the format of the response content is JSON, optional
- `-H &quot;Content-Type: application/json&quot;`: indicate the format of the request body is JSON, optional
- `-H &quot;X-EMC-REST-CLIENT: true&quot;`: to log into the Unity REST API server, required
- `&quot;https://10.245.83.44:443/api/types/pool/instances?compact=True&amp;fields=id,name&quot;`: request URI
  - `https://10.245.83.44:443`: base URI, required
  - `/api/types/pool/instances`: request URI for resource collection query, required
  - `?compact=True`: indicate whether or not to omit metadata for each instance in the response, optional
  - `&amp;fields=id,name`: indicate which attributes of the resource to query, required

It is easy to send the same request using Python:

python

```python
import requests

headers = {
    "Accept": "application/json",
    "Content-type": "application/json",
    "X-EMC-REST-CLIENT": "true"
}

user = "admin"
```

```
passwd = "Password123!"
ipaddr = "10.245.83.44"
port = 443

# Get pool instances
url = "https://{}:{}/api/types/pool/instances?compact=True" \
    "&fields=id,name".format(ipaddr, port)

r = requests.get(url,
            headers=headers,
            auth=(user, passwd),
            verify=False)

print(r.status_code)
print(r.text)
```

In the following chapters, we will learn more about the Unity REST API and how to make more complex requests.

Authentication and Authorization

To make a Unity REST API request, there is something we need to know:

- Every Unity REST API request must be authenticated, except the query of basicSystemInfo resource type
- The Unity REST API uses the standard HTTP Basic authentication mechanism to authenticate the REST requests
- Header `X-EMC-REST-CLIENT: true` is required for logging into Unity REST API server
- Header `EMC-CSRF-TOKEN` is required for POST and DELETE requests
- The following headers should also be included:
  - `Accept: application/json`: to indicate the format of the response content is JSON
  - `Content-type: application/json`: to indicate the format of the request body is JSON, it is required if there is a request body

To get CSRF token, we could make a basic GET request and get the token from the response headers.

Example - Get CSRF token

In this case, we learn how to get CSRF token for POST/DELETE request usage.

First, we send a GET request, in this case, we make a collection query for resource "user":

bash

```
curl -i -k -L \
  -u admin:Password123! \
  -c cookie.txt \
  -H "Accept: application/json" \
  -H "Content-Type: application/json" \
  -H "X-EMC-REST-CLIENT: true" \
  "https://10.245.83.44/api/types/user/instances"
```

This is the response:

```
HTTP/1.1 200 OK
Date: Thu, 27 Aug 2020 01:39:17 GMT
Server: Apache
X-Frame-Options: SAMEORIGIN
Strict-Transport-Security: max-age=63072000; includeSubdomains;
Set-Cookie:
mod_sec_emc=value3&1&value1&iqVsJ%2BQSevmw8FNGWoI%2FkWOzydl0oqhuJNaWfLy3oiq7iwH7fH
ckEcjlZ2GBo5fg%0A&value2&eaae0856a499fcc3b91408224a7307abc203b6684401cf1b912098778c5
79548; path=/; secure; HttpOnly
EMC-CSRF-TOKEN:
A+QAtubhfNa0S6dkPyusYCuuHWnuFfzx7GteLQNEm8KOQgQHe0t90Vp0B09E6V7AwMo90iC0ee5EV
GR0LQrWQcDLs3zLAD1vRThgPyPZWkY=
Pragma: no-cache
Cache-Control: no-cache, no-store, max-age=0
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Content-Language: en-US
Vary: Accept-Encoding
Transfer-Encoding: chunked
Content-Type: application/json; version=1.0;charset=UTF-8

{"@base":"https://10.245.83.44/api/types/user/instances?
per_page=2000","updated":"2020-08-27T01:39:17.471Z","links":[{"rel":"self","href":"&page=1"}],"entries":
[{"@base":"https://10.245.83.44/api/instances/user","updated":"2020-08-27T01:39:17.471Z","links":
[{"rel":"self","href":"/user_admin"}],"content":{"id":"user_admin"}}]}
```

We could get CSRF token from the headers of response, in this case, the CSRF token is `dNrzcvqLkkzcWDNK1O1XBIUzsbldshcss5Jt+mye16D9GuRU+dFTgZN1zGJ1E/jYm6Slo531D9JUdffuY4ViI+aL+MLsIXJ2cwS3T1t24fI=`.

Now, we could use the token to send POST or DELETE request:

bash

```
curl -i -k -L \
  -u admin:Password123! \
  -c cookie.txt \
  -b cookie.txt \
  -H "Accept: application/json" \
  -H "Content-Type: application/json" \
  -H "X-EMC-REST-CLIENT: true" \
  -H "EMC-CSRF-TOKEN:
A+QAtubhfNa0S6dkPyusYCuuHWnuFfzx7GteLQNEm8KOQgQHe0t90Vp0B09E6V7AwMo90iC0ee5EVG
R0LQrWQcDLs3zLAD1vRThgPyPZWkY=" \
  -X DELETE "https://10.245.83.44:443/api/instances/storageResource/sv_19166"
```

How to get attributes of resource

As we discussed in the first call, `fields=id,name` parameter is specified in request URI to tell Unity which attributes we need. But actually most of the resources have a bunch of attributes, not only id and name.

In order to get all attributes of a resource, we could send a GET request to get the resource types.

Example - Get attributes of pool

In this case, we learn how to get all attributes of pool.

We send a GET request to get pool types with `fields=attributes.name` parameter specified:

bash

```bash
curl -i -k -L \
  -u admin:Password123! \
  -c cookie.txt \
  -H "Accept: application/json" \
  -H "Content-Type: application/json" \
  -H "X-EMC-REST-CLIENT: true" \
  "https://10.245.83.44/api/types/pool?compact=True&fields=attributes.name"
```

Unity will return all the attribute names of pool:

json

```json
{
  "content": {
    "name": "pool",
    "attributes": [
      {
        "name": "dataReductionRatio"
      },
      {
        "name": "snapSpaceHarvestLowThreshold"
      },
      {
        "name": "sizeUsed"
      },
      {
        "name": "hasCompressionEnabledLuns"
      },
      {
        "name": "poolSpaceHarvestLowThreshold"
      },
      {
        "name": "sizePreallocated"
      },
      {
        "name": "snapSizeSubscribed"
      },
      {
        "name": "sizeFree"
      },
      {
        "name": "health"
    },
    {
      "name": "name"
    },
    {
      "name": "nonBaseSizeSubscribed"
    },
    {
      "name": "compressionPercent"
    },
    {
      "name": "isAllFlash"
    },
    {
      "name": "snapSpaceHarvestHighThreshold"
    },
    {
      "name": "hasCompressionEnabledFs"
    },
    {
      "name": "alertThreshold"
    },
    {
      "name": "creationTime"
    },
    {
      "name": "id"
    },
    {
      "name": "metadataSizeSubscribed"
    },
    {
      "name": "isSnapHarvestEnabled"
    },
    {
      "name": "isEmpty"
    },
    {
      "name": "dataReductionSizeSaved"
    },
    {
      "name": "hasDataReductionEnabledLuns"
    },
    {
      "name": "compressionRatio"
    },
    {
      "name": "raidType"
    },
    {
      "name": "hasDataReductionEnabledFs"
    },
    {
      "name": "type"
    },
    {
      "name": "snapSizeUsed"
    },
    {
      "name": "harvestState"
    },
    {
      "name": "poolSpaceHarvestHighThreshold"
    },
    {
      "name": "rebalanceProgress"
    },
    {
      "name": "sizeTotal"
    },
    {
      "name": "dataReductionPercent"
    },
    {
      "name": "metadataSizeUsed"
    },
    {
      "name": "nonBaseSizeUsed"
    },
    {
      "name": "tiers"
    },
    {
      "name": "isFASTCacheEnabled"
    },
    {
      "name": "sizeSubscribed"
    },
    {
      "name": "compressionSizeSaved"
    },
    {
      "name": "poolFastVP"
    },
    {
      "name": "description"
    },
    {
      "name": "isHarvestEnabled"
    }
   ]
  }
}
```

Now, we could concatenate all the attributes and specify it in fields parameter to get all attributes for pools:

bash

```
curl -i -k -L \
  -u admin:Password123! \
  -c cookie.txt \
  -H "Accept: application/json" \
  -H "Content-Type: application/json" \
  -H "X-EMC-REST-CLIENT: true" \
  "https://10.245.83.44:443/api/types/pool/instances?
compact=True&fields=dataReductionRatio,snapSpaceHarvestLowThreshold,sizeUsed,hasCompressionEn
abledLuns,poolSpaceHarvestLowThreshold,sizePreallocated,snapSizeSubscribed,sizeFree,health,na
me,nonBaseSizeSubscribed,compressionPercent,isAllFlash,snapSpaceHarvestHighThreshold,hasCompr
essionEnabledFs,alertThreshold,creationTime,id,metadataSizeSubscribed,isSnapHarvestEnabled,isE
mpty,dataReductionSizeSaved,hasDataReductionEnabledLuns,compressionRatio,raidType,hasDataRedu
ctionEnabledFs,type,snapSizeUsed,harvestState,poolSpaceHarvestHighThreshold,rebalanceProgress,
sizeTotal,dataReductionPercent,metadataSizeUsed,nonBaseSizeUsed,tiers,isFASTCacheEnabled,sizeS
ubscribed,compressionSizeSaved,poolFastVP,description,isHarvestEnabled"
```

Unity will return all the attributes of pools, this is the result of one of the pool:

json

```
{
    "content": {
        "id": "pool_1",
        "raidType": 1,
        "harvestState": 0,
        "type": 1,
        "health": {
            "value": 5,
            "descriptionIds": [
                "ALRT_COMPONENT_OK"
            ],
            "descriptions": [
                "The component is operating normally. No action is required."
            ]
        },
        "name": "Cinder_Pool",
        "description": "",
        "sizeFree": 1975148085248,
        "sizeTotal": 2359010787328,
        "sizeUsed": 331662901248,
        "sizePreallocated": 52199800832,
        "compressionSizeSaved": 0,
        "compressionPercent": 0,
        "compressionRatio": 1.0,
        "dataReductionSizeSaved": 0,
        "dataReductionPercent": 0,
        "dataReductionRatio": 1.0,
        "sizeSubscribed": 1154576531456,
        "alertThreshold": 70,
        "hasCompressionEnabledLuns": false,
        "hasCompressionEnabledFs": false,
        "hasDataReductionEnabledLuns": false,
        "hasDataReductionEnabledFs": false,
    "isFASTCacheEnabled": false,
    "tiers": [
      {
        "tierType": 10,
        "raidType": 0,
        "sizeTotal": 0,
        "sizeUsed": 0,
        "sizeFree": 0,
        "sizeMovingDown": 0,
        "sizeMovingUp": 0,
        "sizeMovingWithin": 0,
        "name": "Extreme Performance",
        "diskCount": 0
      },
      {
        "tierType": 20,
        "stripeWidth": 5,
        "raidType": 1,
        "sizeTotal": 2359010787328,
        "sizeUsed": 383862702080,
        "sizeFree": 1975148085248,
        "sizeMovingDown": 0,
        "sizeMovingUp": 0,
        "sizeMovingWithin": 0,
        "name": "Performance",
        "diskCount": 5,
        "poolUnits": [
          {
            "id": "rg_1"
          }
        ]
      },
      {
        "tierType": 30,
        "raidType": 0,
        "sizeTotal": 0,
        "sizeUsed": 0,
        "sizeFree": 0,
        "sizeMovingDown": 0,
        "sizeMovingUp": 0,
        "sizeMovingWithin": 0,
        "name": "Capacity",
        "diskCount": 0
      }
    ],
    "creationTime": "2019-12-13T06:39:37.000Z",
    "isEmpty": false,
    "poolFastVP": {
      "status": 3,
      "relocationRate": 2,
      "type": 2,
      "isScheduleEnabled": true,
      "relocationDurationEstimate": "00:00:00.000",
      "sizeMovingDown": 0,
            "sizeMovingUp": 0,
            "sizeMovingWithin": 0,
            "percentComplete": 0,
            "dataRelocated": 0,
            "lastStartTime": "2020-08-26T22:00:00.000Z",
            "lastEndTime": "2020-08-27T06:00:00.000Z"
        },
        "isHarvestEnabled": true,
        "isSnapHarvestEnabled": false,
        "poolSpaceHarvestHighThreshold": 95.0,
        "poolSpaceHarvestLowThreshold": 85.0,
        "snapSpaceHarvestHighThreshold": 25.0,
        "snapSpaceHarvestLowThreshold": 20.0,
        "metadataSizeSubscribed": 438355099648,
        "snapSizeSubscribed": 139622072320,
        "nonBaseSizeSubscribed": 139622072320,
        "metadataSizeUsed": 314337918976,
        "snapSizeUsed": 35635200,
        "nonBaseSizeUsed": 35635200,
        "isAllFlash": false
    }
},
```

How to query the resource

There are two types of query:

- Collection query: to get all instances of the resource
- Instance query: to get a specific resource instance

From client part of view, the only difference of these two queries is the request URI:

- Request URI of collection query is `/api/types/&lt;resource&gt;/instances`
- Request URI of instance query is `/api/instances/&lt;resource&gt;/&lt;id&gt;` or `/api/instances/&lt;resource&gt;/name:&lt;value&gt;`

Example - Collection query

We already learned collection query in the first call, so we won't introduce it again.

Example - Instance query

In this case, we will learn how to query a specific instance.

We can get the specific pool by pool id:

bash

```bash
curl -i -k -L \
  -u admin:Password123! \
  -c cookie.txt \
  -H "Accept: application/json" \
  -H "Content-Type: application/json" \
  -H "X-EMC-REST-CLIENT: true" \
  "https://10.245.83.44:443/api/instances/pool/pool_1?compact=True&fields=id,name"
```

We can also get the specific pool by pool name:

bash

```bash
curl -i -k -L \
  -u admin:Password123! \
  -c cookie.txt \
  -H "Accept: application/json" \
  -H "Content-Type: application/json" \
  -H "X-EMC-REST-CLIENT: true" \
  "https://10.245.83.44:443/api/instances/pool/name:Cinder_Pool?compact=True&fields=id,name"
```

Unity will return the same response:

json

```json
{
  "content": {
    "id": "pool_1",
    "name": "Cinder_Pool"
  }
}
```

Python implementation:

python

```python
import json

import requests

headers = {
    "Accept": "application/json",
    "Content-type": "application/json",
    "X-EMC-REST-CLIENT": "true"
}

user = "admin"
passwd = "Password123!"
ipaddr = "10.245.83.44"
port = 443

resource = "pool"

# Get pool types
url_types = "https://{}:{}/api/types/{}?compact=True&fields=" \
            "attributes.name".format(ipaddr, port, resource)

r = requests.get(url_types,
                 headers=headers,
                 auth=(user, passwd),
                 verify=False)
```

```
# Concatenate all attributes into one string for resource query
j = json.loads(r.content)
attrs = [x["name"] for x in j["content"]["attributes"]]
str_attrs = ",".join(attrs)

# Get pool instance by id
url = "https://{}:{}/api/instances/{}/pool_1?compact=True&fields" \
      "={}".format(ipaddr, port, resource, str_attrs)
# Get pool instance by name
# url = "https://{}:{}/api/instances/{}/name:Cinder_Pool?compact=True&fields" \
#       "={}".format(ipaddr, port, resource, str_attrs)

r = requests.get(url,
                 headers=headers,
                 auth=(user, passwd),
                 verify=False)

j = json.loads(r.content)
print(j)
```

How to paginate the response data

Pagination is supported by Unity REST API to group and select resource instances in a response.

The following parameters are used for pagination:

- `per_page`: specify how many resource instances to return in one page
- `page`: specify which page to return in a response
- `with_entrycount`: indicate whether or not to return last page and entryCount info

A valid pagination parameter looks like: `per_page=10&amp;page=5&amp;with_entrycount=True`.

Example - Get luns in page 2 (2 luns per page)

In this case, we learn how to direct the Unity REST API server to group 2 lun instances per page and return the instances in page 2.

Send a GET request with pagination parameters specified, the parameter is `per_page=2&amp;page=2&amp;with_entrycount=True`:

bash

```bash
curl -i -k -L \
  -u admin:Password123! \
  -c cookie.txt \
  -H "Accept: application/json" \
  -H "Content-Type: application/json" \
  -H "X-EMC-REST-CLIENT: true" \
  "https://10.245.83.44:443/api/types/lun/instances?compact=True&per_page=2&page=2&with_entrycount=True&fields=id,name"
```

Unity will return 2 lun instances in page 2:

json

```
{
  "@base": "https://10.245.83.44/api/types/lun/instances?
fields=id,name&per_page=2&compact=true",
  "updated": "2020-08-25T07:30:19.910Z",
  "links": [
    {
      "rel": "self",
      "href": "&page=2"
    },
    {
      "rel": "first",
      "href": "&page=1"
    },
    {
      "rel": "prev",
      "href": "&page=1"
    },
    {
      "rel": "next",
      "href": "&page=3"
    },
    {
      "rel": "last",
      "href": "&page=67"
    }
  ],
  "entryCount": 134,
  "entries": [
    {
      "content": {
        "id": "sv_12747",
        "name": "volume-f9945835-28e9-4556-aeba-84e595c14628"
      }
    },
    {
      "content": {
        "id": "sv_12748",
        "name": "volume-9ce2c6b6-6483-4371-ad83-708b776265ca"
      }
    }
  ]
}
```

Python implementation:

python

```
import json

import requests

headers = {
    "Accept": "application/json",
    "Content-type": "application/json",
    "X-EMC-REST-CLIENT": "true"
}

user = "admin"
passwd = "Password123!"
ipaddr = "10.245.83.44"
port = 443

url = "https://{}:{}/api/types/lun/instances?compact=True&per_page=2&page=2" \
      "&with_entrycount=True&fields=id,name".format(ipaddr, port)

r = requests.get(url,
                 headers=headers,
                 auth=(user, passwd),
                 verify=False)

j = json.loads(r.content)
print(j)
```

How to filter the response data

Filtering is supported by Unity REST API to get specific resource instances in a response.

The filter parameter works like a SQL WHERE clause. We could specify a filter expression when querying instances, Unity will return the matched instances.

The `filter=&lt;expression&gt;` parameter is used for filtering. A valid filtering parameter looks like: `filter=severity eq 3`.

Unity supports varieties of expressions, please refer Unity REST API Programmer's Guide for more details.

Example - Get pool by id

In this case, we learn how to get specific instances.

Send a GET request with filtering parameter specified, the parameter is `filter=id eq &quot;pool_1&quot;`, in curl, the URL is required to be encoded, so we encode it to `filter=id%20eq%20%22pool_1%22`:

bash

```
curl -i -k -L \
  -u admin:Password123! \
  -c cookie.txt \
  -H "Accept: application/json" \
  -H "Content-Type: application/json" \
  -H "X-EMC-REST-CLIENT: true" \
  "https://10.245.83.44:443/api/types/pool/instances?
compact=True&filter=id%20eq%20%22pool_1%22&fields=id,name"
```

Unity will return the pool which id equal to "pool_1":

json

```
{
  "@base": "https://10.245.83.44/api/types/pool/instances?filter=id eq
\"pool_1\"&fields=id,name&per_page=2000&compact=true",
  "updated": "2020-08-25T08:11:58.657Z",
  "links": [
    {
      "rel": "self",
      "href": "&page=1"
    }
  ],
  "entries": [
    {
      "content": {
        "id": "pool_1",
        "name": "Cinder_Pool"
      }
    }
  ]
}
```

Python implementation:

python

```python
import json

import requests

headers = {
    "Accept": "application/json",
    "Content-type": "application/json",
    "X-EMC-REST-CLIENT": "true"
}

user = "admin"
passwd = "Password123!"
ipaddr = "10.245.83.44"
port = 443

url = 'https://{}:{}/api/types/pool/instances?compact=True' \
      '&filter=id eq "pool_1"&fields=id,name'.format(ipaddr, port)

r = requests.get(url,
                 headers=headers,
                 auth=(user, passwd),
                 verify=False)

j = json.loads(r.content)
print(j)
```

How to sort the response data

Sorting is supported by Unity REST API to sort resource instances in a response.

The following parameters are used for sorting:

- `orderby`: specify which attribute the response data will order by
- `asc`: (default) sort the response data in ascending order
- `desc`: sort the response data in descending order

A valid sorting parameter looks like: `orderby=name desc`.

Example - Sort pools by name

In this case, we learn how to sort pools by name.

Send a GET request with sorting parameter specified, the parameter is `orderby=name desc`, in curl, the URL is required to be encoded, so we encode it to `orderby=name%20desc`:

bash

```bash
curl -i -k -L \
  -u admin:Password123! \
  -c cookie.txt \
  -H "Accept: application/json" \
  -H "Content-Type: application/json" \
  -H "X-EMC-REST-CLIENT: true" \
  "https://10.245.83.44:443/api/types/pool/instances?
compact=True&orderby=name%20desc&fields=id,name"
```

Unity will return the pools which ordered by name in descending order:

json

```json
{
  "@base": "https://10.245.83.44/api/types/pool/instances?fields=id,name&orderby=name
desc&per_page=2000&compact=true",
  "updated": "2020-08-25T08:37:12.790Z",
  "links": [
    {
      "rel": "self",
      "href": "&page=1"
    }
  ],
  "entries": [
    {
      "content": {
        "id": "pool_2",
        "name": "Manila_Pool"
      }
    },
    {
      "content": {
        "id": "pool_3",
        "name": "Flash_Pool"
      }
    },
    {
      "content": {
        "id": "pool_1",
        "name": "Cinder_Pool"
      }
    }
  ]
}
```

Python implementation:

python

```
import json

import requests

headers = {
    "Accept": "application/json",
    "Content-type": "application/json",
    "X-EMC-REST-CLIENT": "true"
}

user = "admin"
passwd = "Password123!"
ipaddr = "10.245.83.44"
port = 443

url = "https://{}:{}/api/types/pool/instances?compact=True&orderby=name desc" \
      "&fields=id,name".format(ipaddr, port)

r = requests.get(url,
                 headers=headers,
                 auth=(user, passwd),
                 verify=False)

j = json.loads(r.content)
print(j)
```

How to aggregate the response data

Aggregation is supported by Unity REST API to group specific resource instances in a response.

The `groupby` parameter is used for aggregation, and works like a SQL Group By clause. A valid aggregation parameter looks like: `groupby=type`.

Example - Aggregate pools by type

In this case, we learn how to group pools by type, and how to define new attributes we need.

Send a GET request with aggregation and fields parameter specified. The aggregation parameter is `groupby=type`. The fields parameter is `fields=type,total::@sum(sizeTotal),free::@sum(sizeFree)` which two new attributes "total" and "free" are defined to calculate the sum of sizeTotal and sizeFree of the pools in the group (we will discuss the new attribute definition in next chapter), the request is:

bash

```
curl -i -k -L \
  -u admin:Password123! \
  -c cookie.txt \
  -H "Accept: application/json" \
  -H "Content-Type: application/json" \
  -H "X-EMC-REST-CLIENT: true" \
  "https://10.245.83.44:443/api/types/pool/instances?
compact=True&fields=type,total::@sum(sizeTotal),free::@sum(sizeFree)&groupby=type"
```

Unity will group the type 1 pools, calculate and return the sum of sizeTotal and sizeFree of the pools:

json

```json
{
  "@base": "https://10.245.83.44/api/types/pool/instances?
fields=type,total::@sum(sizeTotal),free::@sum(sizeFree)&groupby=type&per_page=2000&compact=tru
e",
  "updated": "2020-08-28T03:36:07.744Z",
  "links": [
    {
      "rel": "self",
      "href": "&page=1"
    }
  ],
  "entries": [
    {
      "content": {
        "type": 1,
        "total": 5829881233408,
        "free": 5320927608832
      }
    }
  ]
}
```

Python implementation:

python

```python
import json

import requests

headers = {
    "Accept": "application/json",
    "Content-type": "application/json",
    "X-EMC-REST-CLIENT": "true"
}

user = 'admin'
passwd = 'Password123!'
ipaddr = '10.245.83.44'
port = 443
```

```
url = 'https://{}:{}/api/types/pool/instances?compact=True&' \
    'fields=type,total::@sum(sizeTotal),free::@sum(sizeFree)' \
    '&groupby=type'.format(ipaddr, port)

r = requests.get(url,
          headers=headers,
          auth=(user, passwd),
          verify=False)

j = json.loads(r.content)
print(j)
```

How to define new attributes from existing attributes

Unity REST API provides an easy to use but powerful way to define new attributes from an expression associated with one or more existing attributes. The new attributes could be used to filter or sort the response data.

The new attributes are defined in `fields` parameter, the syntax is `fields=&lt;new_attr_name&gt;::&lt;new_attr_expr&gt;,&lt;attr1&gt;,&lt;attr2&gt;`. Unity REST API provides varieties of expression types including arithmetic expression, conditional expression, etc.. Please refer Unity REST API Programmer's Guide for more details.

A valid expression looks like: `fields=type,total::@sum(sizeTotal)`.

Example - Define new attribute using arithmetic expression

In this case, we learn how to define new attribute using arithmetic expression.

Send a GET request with arithmetic expression `percent::sizeUsed*100/sizeTotal` defined in fields parameter:

bash

```
curl -i -k -L \
  -u admin:Password123! \
  -c cookie.txt \
  -H "Accept: application/json" \
  -H "Content-Type: application/json" \
  -H "X-EMC-REST-CLIENT: true" \
  "https://10.245.83.44:443/api/types/pool/instances?
compact=True&fields=id,name,sizeUsed,sizeTotal,percent::sizeUsed*100/sizeTotal"
```

Unity will return all the pool instances with new attribute "percent" whose value is calculated by `sizeUsed*100/sizeTotal`:

json

```
{
  "@base": "https://10.245.83.44/api/types/pool/instances?
fields=id,name,sizeUsed,sizeTotal,percent::sizeUsed*100/sizeTotal&per_page=2000&compact=true",
  "updated": "2020-08-28T06:04:54.480Z",
  "links": [
    {
      "rel": "self",
      "href": "&page=1"
    }
  ],
  "entries": [
    {
      "content": {
        "id": "pool_1",
        "name": "Cinder_Pool",
        "sizeTotal": 2359010787328,
        "sizeUsed": 331662901248,
        "percent": 14.059406
      }
    },
    {
      "content": {
        "id": "pool_2",
        "name": "Manila_Pool",
        "sizeTotal": 1899986157568,
        "sizeUsed": 43916820480,
        "percent": 2.3114285
      }
    },
    {
      "content": {
        "id": "pool_3",
        "name": "Flash_Pool",
        "sizeTotal": 1570884288512,
        "sizeUsed": 67645734912,
        "percent": 4.30622
      }
    }
  ]
}
```

Python implementation:

python

```python
import json

import requests

headers = {
    "Accept": "application/json",
    "Content-type": "application/json",
    "X-EMC-REST-CLIENT": "true"
}

user = 'admin'
passwd = 'Password123!'
ipaddr = "10.245.83.44"
port = 443

# Get pools with new attribute "percent" which calculated by sizeUsed and
# sizeTotal.
url = 'https://{}:{}/api/types/pool/instances?compact=True&fields=id,name,' \
      'sizeUsed,sizeTotal,percent::sizeUsed*100/sizeTotal'.format(ipaddr, port)

r = requests.get(url,
                 headers=headers,
                 auth=(user, passwd),
                 verify=False)

j = json.loads(r.content)
print(j)
```

Example - Define new attribute using conditional expression

In this case, we learn how to define new attribute using conditional expression.

Send a GET request with conditional expression `newName::@concat(name,type)` defined in fields parameter:

bash

```bash
curl -i -k -L \
  -u admin:Password123! \
  -c cookie.txt \
  -H "Accept: application/json" \
  -H "Content-Type: application/json" \
  -H "X-EMC-REST-CLIENT: true" \
  "https://10.245.83.44:443/api/types/pool/instances?
compact=True&fields=id,name,type,newName::@concat(name,type)"
```

Unity will return all the pool instances with new attribute "newName" which concatenated by name and type:

json

```json
{
  "@base": "https://10.245.83.44/api/types/pool/instances?
fields=id,name,type,newName::@concat(name,type)&per_page=2000&compact=true",
  "updated": "2020-08-28T06:05:35.286Z",
  "links": [
    {
      "rel": "self",
      "href": "&page=1"
    }
  ],
  "entries": [
    {
      "content": {
        "id": "pool_1",
        "type": 1,
        "name": "Cinder_Pool",
        "newName": "Cinder_Pool1"
      }
    },
    {
      "content": {
        "id": "pool_2",
        "type": 1,
        "name": "Manila_Pool",
        "newName": "Manila_Pool1"
      }
    },
    {
      "content": {
        "id": "pool_3",
        "type": 1,
        "name": "Flash_Pool",
        "newName": "Flash_Pool1"
      }
    }
  ]
}
```

Python implementation:

python

```python
import json

import requests

headers = {
    "Accept": "application/json",
    "Content-type": "application/json",
    "X-EMC-REST-CLIENT": "true"
}

user = "admin"
passwd = "Password123!"
ipaddr = "10.245.83.44"
port = 443

# Get luns with new attribute "newName" which concatenate by name and type.
url = "https://{}:{}/api/types/storageResource/instances?compact=True" \
      "&fields=id,name,type,newName::@concat(name,type)".format(ipaddr, port)

r = requests.get(url,
                 headers=headers,
                 auth=(user, passwd),
                 verify=False)

j = json.loads(r.content)
print(j)
```

How to create a new resource

To create a new resource, we need to follow these steps:

- Get CSRF token which used for authentication and authorization, it is required
- Prepare resource creation data
- Send a POST request with CSRF token and resource creation data specified

For the details of the resource creation data, please refer the API references.

Example - Create a new lun

In this case, we will create a new lun in pool "pool_1" with name "test_lun_1" and size 1GB.

First, we need to get CSRF token by sending a GET request, and save the token for upcoming usage:

bash

```bash
token=$(curl -i -k -L \
  -u admin:Password123! \
  -c cookie.txt \
  -H "Accept: application/json" \
  -H "Content-Type: application/json" \
  -H "X-EMC-REST-CLIENT: true" \
  "https://10.245.83.44/api/types/user/instances" | grep EMC-CSRF-TOKEN)

token=`echo $token | sed 's/\r//g'`
```

Then we send a POST request with CSRF token and lun creation data specified:

bash

```bash
curl -i -k -L \
  -u admin:Password123! \
  -c cookie.txt \
  -b cookie.txt \
  -H "Accept: application/json" \
  -H "Content-Type: application/json" \
  -H "X-EMC-REST-CLIENT: true" \
  -H "${token}" \
  -d "{\"name\": \"test_lun_1\", \"lunParameters\": {\"pool\": {\"id\": \"pool_1\"}, \"size\": 1073741824}}" \
  -X POST "https://10.245.83.44:443/api/types/storageResource/action/createLun"
```

Unity will return the id of the new created lun:

json

```json
{
  "@base": "https://10.245.83.44/api/types/storageResource/action/createLun",
  "updated": "2020-08-26T06:27:31.524Z",
  "links": [
    {
      "rel": "self",
      "href": "/sv_19166"
    }
  ],
  "content": {
    "storageResource": {
      "id": "sv_19166"
    }
  }
}
```

Python implementation:

python

```
import json

import requests

headers = {
    "Accept": "application/json",
    "Content-type": "application/json",
    "X-EMC-REST-CLIENT": "true"
}

user = "admin"
passwd = "Password123!"
ipaddr = "10.245.83.44"
port = 443

# New a session
session = requests.Session()

# URL used to get CSRF token
url = "https://{}:{}/api/types/user/instances".format(ipaddr, port)

r = session.get(url,
                headers=headers,
                auth=(user, passwd),
                verify=False)

# Update CSRF token
headers["EMC-CSRF-TOKEN"] = r.headers["EMC-CSRF-TOKEN"]

# Data used to create Lun
data = {
    "name": "test_lun_1",
    "lunParameters": {
        "pool": {
            "id": "pool_1"
        },
        "size": 1024 * 1024 * 1024
    }
}

# URL used to create Lun
url_create = "https://{}:{}/api/types/storageResource/action/createLun" \
             "".format(ipaddr, port)

# Create Lun
```

```
r = session.request('POST',
                    url_create,
                    headers=headers,
                    data=json.dumps(data),
                    verify=False)

j = json.loads(r.content)
print(j)
```

How to modify a resource

To modify a resource, we need to follow these steps:

- Get CSRF token which used for authentication and authorization, it is required
- Prepare resource modification data
- Send a POST request with CSRF token and resource modification data specified

For the details of the resource modification data, please refer the API references.

Example - Modify a lun

In this case, we will modify the description and size for an existing lun.

First, we need to get CSRF token by sending a GET request, and save the token for upcoming usage:

bash
```
token=$(curl -i -k -L \
  -u admin:Password123! \
  -c cookie.txt \
  -H "Accept: application/json" \
  -H "Content-Type: application/json" \
  -H "X-EMC-REST-CLIENT: true" \
  "https://10.245.83.44/api/types/user/instances" | grep EMC-CSRF-TOKEN)

token=`echo $token | sed 's/\r//g'`
```

Then we send a POST request with CSRF token and lun modification data specified:

bash
```
curl -i -k -L \
  -u admin:Password123! \
  -c cookie.txt \
  -b cookie.txt \
  -H "Accept: application/json" \
  -H "Content-Type: application/json" \
  -H "X-EMC-REST-CLIENT: true" \
  -H "${token}" \
  -d "{\"description\": \"Modified Lun.\", \"lunParameters\": {\"size\": 3221225472}}" \
  -X POST "https://10.245.83.44:443/api/instances/storageResource/sv_19166/action/modifyLun"
```

Unity will return 204 No Content:

json

```
HTTP/1.1 204 No Content
Date: Wed, 26 Aug 2020 06:51:07 GMT
Server: Apache
X-Frame-Options: SAMEORIGIN
Strict-Transport-Security: max-age=63072000; includeSubdomains;
EMC-CSRF-TOKEN:
NIvFylcvG5vysYsT7WxCh93dJEXNstHXrontaVRkbVCDaWyd0oB3rhfKNMNSpEtKicMk/
BJS+Ho7LtA1PtjnGS1ZHopR09vlnavq+V9wIAw=
Content-Length: 0
```

Python implementation:

python

```python
import json

import requests

headers = {
    "Accept": "application/json",
    "Content-type": "application/json",
    "X-EMC-REST-CLIENT": "true"
}

user = "admin"
passwd = "Password123!"
ipaddr = "10.245.83.44"
port = 443

# New a session
session = requests.Session()

# URL used to get CSRF token
url = "https://{}:{}/api/types/user/instances".format(ipaddr, port)

r = session.get(url,
                headers=headers,
                auth=(user, passwd),
                verify=False)

# Update CSRF token
headers["EMC-CSRF-TOKEN"] = r.headers["EMC-CSRF-TOKEN"]

# Data used to modify Lun
data = {
    "description": "Modified Lun.",
    "lunParameters": {
        "size": 3 * 1024 * 1024 * 1024
    }
}
```

```
lun_id = 'sv_19151'

# URL used to modify Lun
url_modify = "https://{}:{}/api/instances/storageResource/{}/action/" \
            "modifyLun".format(ipaddr, port, lun_id)

# Modify Lun
r = session.request('POST',
                   url_modify,
                   headers=headers,
                   data=json.dumps(data),
                   verify=False)

print(r.status_code)
```

How to delete a resource

To delete a resource, we need to follow these steps:

- Get CSRF token which used for authentication and authorization, it is required
- Send a DELETE request with CSRF token specified

Example - Delete a lun

In this case, we will delete an existing lun.

First, we need to get CSRF token by sending a GET request, and save the token for upcoming usage:

bash

```bash
token=$(curl -i -k -L \
  -u admin:Password123! \
  -c cookie.txt \
  -H "Accept: application/json" \
  -H "Content-Type: application/json" \
  -H "X-EMC-REST-CLIENT: true" \
  "https://10.245.83.44/api/types/user/instances" | grep EMC-CSRF-TOKEN)

token=`echo $token | sed 's/\r//g'`
```

Then we send a DELETE request with CSRF token specified:

bash

```bash
curl -i -k -L \
  -u admin:Password123! \
  -c cookie.txt \
  -b cookie.txt \
  -H "Accept: application/json" \
  -H "Content-Type: application/json" \
  -H "X-EMC-REST-CLIENT: true" \
  -H "${token}" \
  -X DELETE "https://10.245.83.44:443/api/instances/storageResource/sv_19166"
```

Unity will return 204 No Content:

json

```
HTTP/1.1 204 No Content
Date: Wed, 26 Aug 2020 07:01:01 GMT
Server: Apache
X-Frame-Options: SAMEORIGIN
Strict-Transport-Security: max-age=63072000; includeSubdomains;
EMC-CSRF-TOKEN:
NIvFylcvG5vysYsT7WxCh93dJEXNstHXrontaVRkbVCDaWyd0oB3rhfKNMNSpEtKicMk/
BJS+Ho7LtA1PtjnGS1ZHopR09vlnavq+V9wIAw=
Content-Length: 0
```

Python implementation:

python

```python
import requests

headers = {
   "Accept": "application/json",
   "Content-type": "application/json",
   "X-EMC-REST-CLIENT": "true"
}

user = "admin"
passwd = "Password123!"
ipaddr = "10.245.83.44"
port = 443

# New a session
session = requests.Session()

# URL used to get CSRF token
url = "https://{}:{}/api/types/user/instances".format(ipaddr, port)

r = session.get(url,
             headers=headers,
             auth=(user, passwd),
             verify=False)

# Update CSRF token
headers["EMC-CSRF-TOKEN"] = r.headers["EMC-CSRF-TOKEN"]

lun_id = 'sv_19151'

# URL used to delete Lun
url_delete = "https://{}:{}/api/instances/storageResource/{}".format(ipaddr,
                                                                   port,
                                                                   lun_id)

# Delete Lun
r = session.request('DELETE',
               url_delete,
               headers=headers,
               verify=False)

print(r.status_code)
```

How to make an asynchronous request

By default, all Unity REST API requests are synchronous, that means the client/server connection keep opening until the request completed.

To send an asynchronous request, we could specify timeout=0 as parameter in request URL, Unity will return a job id immediately, we could use it to get the job status periodically.

Example - Create a lun in asynchronous mode

In this case, we will create a new lun in asynchronous mode.

First, we need to get CSRF token by sending a GET request, and save the token for upcoming usage:

bash

```bash
token=$(curl -i -k -L \
  -u admin:Password123! \
  -c cookie.txt \
  -H "Accept: application/json" \
  -H "Content-Type: application/json" \
  -H "X-EMC-REST-CLIENT: true" \
  "https://10.245.83.44/api/types/user/instances" | grep EMC-CSRF-TOKEN)

token=`echo $token | sed 's/\r//g'`
```

Then we send a POST request with CSRF token, lun creation data and timeout=0 parameter specified:

bash

```bash
resp=$(curl -i -k -L \
  -u admin:Password123! \
  -c cookie.txt \
  -b cookie.txt \
  -H "Accept: application/json" \
  -H "Content-Type: application/json" \
  -H "X-EMC-REST-CLIENT: true" \
  -H "${token}" \
  -d "{\"name\": \"test_lun_1\", \"lunParameters\": {\"pool\": {\"id\": \"pool_1\"}, \"size\": 1073741824}}" \
  -X POST "https://10.245.83.44:443/api/types/storageResource/action/createLun?timeout=0")
```

Unity will return the new created job:

json

```json
{
  "id": "N-49212",
  "state": 2,
  "instanceId": "root/emc:EMC_UEM_TransactionJobLeaf%InstanceID=N-49212",
  "description": "job.storagevolumeservice.job.Create",
  "stateChangeTime": "2020-08-26T07:31:27.206Z",
  "submitTime": "2020-08-26T07:31:27.108Z",
  "startTime": "2020-08-26T07:31:27.221Z",
  "elapsedTime": "00:00:00.023",
  "estRemainTime": "00:00:03.000",
  "progressPct": 0,
  "tasks": [
   {
     "state": 1,
     "name": "job.blockservice.task.CreateStorageVolume"
   }
  ],
  "owner": "System",
  "requestId": "",
  "methodName": "storageResource.createLun",
  "parametersOut": {},
  "isJobCancelable": false,
  "isJobCancelled": false,
  "clientData": ""
}
```

If Unity not return error, get the job id:

bash

```bash
if [[ $resp == *error* ]];
then
  echo "Job failure."
  exit 1
fi

job_id=$(echo $resp | sed 's/\(.*\)"id":"\(.*\)"\,"state"\(.*\)/\2/')
```

Wait for a moment and check the job status:

bash

```bash
sleep 5

curl -i -k -L \
  -u admin:Password123! \
  -c cookie.txt \
  -H "Accept: application/json" \
  -H "Content-Type: application/json" \
  -H "X-EMC-REST-CLIENT: true" \
  "https://10.245.83.44:443/api/instances/job/${job_id}?compact=True&fields=id,state,progressPct"
```

Unity return the job status:

json

```json
{
  "content": {
    "id": "N-49212",
    "state": 4,
    "progressPct": 100
  }
}
```

Python implementation:

python

```
import json
import time

import requests

headers = {
    "Accept": "application/json",
    "Content-type": "application/json",
    "X-EMC-REST-CLIENT": "true"
}

user = "admin"
passwd = "Password123!"
ipaddr = "10.245.83.44"
port = 443

# New a session
session = requests.Session()

# URL used to get CSRF token
url = "https://{}:{}/api/types/user/instances".format(ipaddr, port)

r = session.get(url,
                headers=headers,
                auth=(user, passwd),
                verify=False)

# Update CSRF token
headers["EMC-CSRF-TOKEN"] = r.headers["EMC-CSRF-TOKEN"]

# Data used to create Lun
data = {
    "name": "test_lun_1",
    "lunParameters": {
        "pool": {
            "id": "pool_1"
        },
        "size": 1024 * 1024 * 1024
    }
}

# URL used to create Lun
url_create = "https://{}:{}/api/types/storageResource/action/createLun" \
             "?timeout=0".format(ipaddr, port)
```

```
# Create Lun
r = session.request('POST',
                    url_create,
                    headers=headers,
                    data=json.dumps(data),
                    verify=False)

j = json.loads(r.content)
print(j)

job_id = j['id']

# Wait for job to finish
time.sleep(5)

# URL used to get job
url = "https://{}:{}/api/instances/job/{}?compact=True" \
      "&fields=id,state,progressPct".format(ipaddr, port, job_id)

# Get job
r = session.get(url,
                headers=headers,
                auth=(user, passwd),
                verify=False)

j = json.loads(r.content)
print(j)
```

How to aggregate requests

We could group non-GET requests into one aggregated request.

To create an aggregated request, we need to create a job instance, within the job instance, create one embedded jobTask instance for each POST request.

Follow these steps:

- Get CSRF token which used for authentication and authorization, it is required
- Prepare job creation data
- Send a POST request with CSRF token and job creation data specified

Example - Create a new lun, a new host and a new hostInitiator

In this case, we will group the creation of lun, host and hostInitiator into one request.

First, we save the job creation data to a json file, with lun, host and hostInitiator creation data contained:

json

```json
{
    "description": "CREATE_LUN_AND_HOST_JOB_DESC",
    "tasks": [
        {
            "name": "CreatingLun",
            "object": "storageResource",
            "action": "createLun",
            "description": "CREATE_LUN_TASK_DESCRIPTION",
            "descriptionArg": "test_lun_1",
            "parametersIn": {
                "name": "test_lun_1",
                "lunParameters": {
                    "pool": {
                        "id": "pool_1"
                    },
                    "size": 1073741824
                }
            }
        },
        {
            "name": "CreatingHost",
            "object": "host",
            "action": "create",
            "description": "CREATE_HOST_TASK_DESCRIPTION",
            "descriptionArg": "test_host_1",
            "parametersIn": {
                "name": "test_host_1",
                "type": 1
            }
        },
        {
            "name": "CreateAndRegisteriSCSIInitiator_0",
            "object": "hostInitiator",
            "action": "create",
            "description": "CREATE_ISCSI_INITIATOR_TASK_DESC",
            "descriptionArg": "iqn.1994-05.com.redhat:2bfbc0884dc4",
            "parametersIn": {
                "host": "@CreatingHost.id",
                "initiatorType": 2,
                "initiatorWWNorIqn": "iqn.1994-05.com.redhat:2bfbc0884dc4"
            }
        }
    ]
}
```

Then we get CSRF token by sending a GET request, and save the token for upcoming usage:

bash

```bash
token=$(curl -i -k -L \
  -u admin:Password123! \
  -c cookie.txt \
  -H "Accept: application/json" \
  -H "Content-Type: application/json" \
  -H "X-EMC-REST-CLIENT: true" \
  "https://10.245.83.44/api/types/user/instances" | grep EMC-CSRF-TOKEN)
```

```
token=`echo $token | sed 's/\r//g'`
```

Finally, we send a POST request with CSRF token, job creation data and timeout=0 parameter specified, in this case, we load the data from the json file:

bash

```
curl -i -k -L \
  -u admin:Password123! \
  -c cookie.txt \
  -b cookie.txt \
  -H "Accept: application/json" \
  -H "Content-Type: application/json" \
  -H "X-EMC-REST-CLIENT: true" \
  -H "${token}" \
  -d "$(sed ':a;N;$!ba;s/\n//g;s/\s*//g' 14_aggregated_request_data.json)" \
  -X POST "https://10.245.83.44:443/api/types/job/instances?
compact=true&visibility=Engineering&timeout=0"
```

Unity will return the new create job:

json

```
{
  "id": "B-161",
  "state": 2,
  "description": "CREATE_LUN_AND_HOST_JOB_DESC",
  "stateChangeTime": "2020-08-26T08:02:12.630Z",
  "submitTime": "2020-08-26T08:02:12.630Z",
  "startTime": "2020-08-26T08:02:12.634Z",
  "progressPct": 0,
  "methodName": "job.create"
}
```

Python implementation:

python

```
import json

import requests

headers = {
    "Accept": "application/json",
    "Content-type": "application/json",
    "X-EMC-REST-CLIENT": "true"
}

user = "admin"
passwd = "Password123!"
ipaddr = "10.245.83.44"
port = 443

# New a session
session = requests.Session()
```

```
# URL used to get CSRF token
url = "https://{}:{}/api/types/user/instances".format(ipaddr, port)

r = session.get(url,
            headers=headers,
            auth=(user, passwd),
            verify=False)

# Update CSRF token
headers["EMC-CSRF-TOKEN"] = r.headers["EMC-CSRF-TOKEN"]

# Data used to create Job
lun_name = "test_lun_1"
pool_id = "pool_1"
host_name = "test_host_1"
initiator_iqn = "iqn.1994-05.com.redhat:2bfbc0884dc4"

data = {
   "description": "CREATE_LUN_AND_HOST_JOB_DESC",
   "tasks": [
      {
         "name": "CreatingLun",
         "object": "storageResource",
         "action": "createLun",
         "description": "CREATE_LUN_TASK_DESCRIPTION",
         "descriptionArg": lun_name,
         "parametersIn": {
            "name": lun_name,
            "lunParameters": {
               "pool": {
                  "id": pool_id
               },
               "size": 1024 * 1024 * 1024
            }
         }
      },
      {
         "name": "CreatingHost",
         "object": "host",
         "action": "create",
         "description": "CREATE_HOST_TASK_DESCRIPTION",
         "descriptionArg": host_name,
         "parametersIn": {
            "name": host_name,
            "type": 1
         }
      },
      {
         "name": "CreateAndRegisteriSCSIInitiator_0",
         "object": "hostInitiator",
         "action": "create",
         "description": "CREATE_ISCSI_INITIATOR_TASK_DESC",
         "descriptionArg": initiator_iqn,
            "parametersIn": {
                "host": "@CreatingHost.id",
                "initiatorType": 2,
                "initiatorWWNorIqn": initiator_iqn
            }
        }
    ]
}

resource = 'job'

# URL used to create Job
url_create = "https://{}:{}/api/types/{}/instances?compact=true" \
             "&visibility=Engineering&timeout=0".format(ipaddr,
                                                        port,
                                                        resource)

# Create Job
r = session.request('POST',
                    url_create,
                    headers=headers,
                    data=json.dumps(data),
                    verify=False)

j = json.loads(r.content)
print(j)
```

Use Cases

Next, we will act as an end user, to learn a more "end-to-end" use case: how to access a new lun in a server via iSCSI.

In order to access the lun in a server, we need to create a host and a hostInitiator in Unity, we also need to configure the lun to allow the access from host, then we could use iSCSI tools to discover, login and access the lun in the server.

In the following introduction, we will cover the steps of:

1. Create a new pool

2. Create a new lun

3. Create a new host

4. Create a new host initiator

5. Modify lun to allow the access from host

The operations in server won't be covered:

1. iSCSI discovery

2. iSCSI login

3. Make file system

4. Access the file system

Create a pool

The pool creation data as below:

json
```json
{
   "name": "test_pool_1",
   "addRaidGroupParameters": [
      {
         "dskGroup": {
            "id": "dg_16"
         },
         "numDisks": 2,
         "raidType": 7,
         "stripeWidth": 2
      }
   ],
   "type": 1,
   "isFASTCacheEnabled": "False"
}
```

Send a POST request to create a new pool:

bash
```bash
token=$(curl -i -k -L \
  -u admin:Password123! \
  -c cookie.txt \
  -H "Accept: application/json" \
  -H "Content-Type: application/json" \
  -H "X-EMC-REST-CLIENT: true" \
  "https://10.229.114.32/api/types/user/instances" | grep EMC-CSRF-TOKEN)

token=`echo $token | sed 's/\r//g'`

curl -i -k -L \
  -u admin:Password123! \
  -c cookie.txt \
  -b cookie.txt \
  -H "Accept: application/json" \
  -H "Content-Type: application/json" \
  -H "X-EMC-REST-CLIENT: true" \
  -H "${token}" \
  -d "$(sed ':a;N;$!ba;s/\n//g;s/\s*//g' 15_use_case_data_pool_create.json)" \
  -X POST "https://10.229.114.32/api/types/pool/instances"
```

Create a lun

The lun creation data as below, please replace the pool id with the new created pool:

json
```json
{
    "name": "test_lun_1",
    "lunParameters": {
        "pool": {
            "id": "pool_8"
        },
        "size": 1073741824
    }
}
```

Send a POST request to create a new lun:

bash

```bash
token=$(curl -i -k -L \
  -u admin:Password123! \
  -c cookie.txt \
  -H "Accept: application/json" \
  -H "Content-Type: application/json" \
  -H "X-EMC-REST-CLIENT: true" \
  "https://10.229.114.32/api/types/user/instances" | grep EMC-CSRF-TOKEN)

token=`echo $token | sed 's/\r//g'`

curl -i -k -L \
  -u admin:Password123! \
  -c cookie.txt \
  -b cookie.txt \
  -H "Accept: application/json" \
  -H "Content-Type: application/json" \
  -H "X-EMC-REST-CLIENT: true" \
  -H "${token}" \
  -d "$(sed ':a;N;$!ba;s/\n//g;s/\s*//g' 15_use_case_data_lun_create.json)" \
  -X POST "https://10.229.114.32/api/types/storageResource/action/createLun"
```

Create a host

Send a POST request to create a new host:

bash

```bash
token=$(curl -i -k -L \
  -u admin:Password123! \
  -c cookie.txt \
  -H "Accept: application/json" \
  -H "Content-Type: application/json" \
  -H "X-EMC-REST-CLIENT: true" \
  "https://10.229.114.32/api/types/user/instances" | grep EMC-CSRF-TOKEN)

token=`echo $token | sed 's/\r//g'`

curl -i -k -L \
  -u admin:Password123! \
  -c cookie.txt \
  -b cookie.txt \
  -H "Accept: application/json" \
  -H "Content-Type: application/json" \
  -H "X-EMC-REST-CLIENT: true" \
  -H "${token}" \
  -d "{\"name\": \"test_host_1\", \"type\": 1}" \
  -X POST "https://10.229.114.32/api/types/host/instances"
```

Create a host initiator

The hostInitiator creation data as below, please replace the host id with the new created host:

json

```json
{
    "host": {"id": "Host_20"},
    "initiatorType": 2,
    "initiatorWWNorIqn": "iqn.1994-05.com.redhat:2bfbc0884dc4"
}
```

Send a POST request to create a new hostInitiator:

bash

```bash
token=$(curl -i -k -L \
  -u admin:Password123! \
  -c cookie.txt \
  -H "Accept: application/json" \
  -H "Content-Type: application/json" \
  -H "X-EMC-REST-CLIENT: true" \
  "https://10.229.114.32/api/types/user/instances" | grep EMC-CSRF-TOKEN)

token=`echo $token | sed 's/\r//g'`

curl -i -k -L \
  -u admin:Password123! \
  -c cookie.txt \
  -b cookie.txt \
  -H "Accept: application/json" \
  -H "Content-Type: application/json" \
  -H "X-EMC-REST-CLIENT: true" \
  -H "${token}" \
  -d "$(sed ':a;N;$!ba;s/\n//g;s/\s*//g' 15_use_case_data_host_initiator.json)" \
  -X POST "https://10.229.114.32/api/types/hostInitiator/instances"
```

Modify lun to allow the access

The lun modification data as below, please replace the host id with the new created host:

json

```json
{
    "lunParameters": {
        "hostAccess": [
            {
                "accessMask": 4,
                "host": {
```

```
                "id": "Host_20"
            }
        }
    ]
  }
}
```

Send a POST request to modify the lun to allow the access from the host:

bash

```bash
token=$(curl -i -k -L \
  -u admin:Password123! \
  -c cookie.txt \
  -H "Accept: application/json" \
  -H "Content-Type: application/json" \
  -H "X-EMC-REST-CLIENT: true" \
  "https://10.229.114.32/api/types/user/instances" | grep EMC-CSRF-TOKEN)

token=`echo $token | sed 's/\r//g'`

curl -i -k -L \
  -u admin:Password123! \
  -c cookie.txt \
  -b cookie.txt \
  -H "Accept: application/json" \
  -H "Content-Type: application/json" \
  -H "X-EMC-REST-CLIENT: true" \
  -H "${token}" \
  -d "$(sed ':a;N;$!ba;s/\n//g;s/\s*//g' 15_use_case_data_lun_modify.json)" \
  -X POST "https://10.229.114.32:443/api/instances/storageResource/sv_745/action/modifyLun?
compact=true&visibility=Engineering&timeout=5"
```

Python implementation

It is easy to implement the use case by Python:

python

```python
import json

import requests


class UnityRestClient:
    def __init__(self, user, password, ipaddr, port=443):
        self.user = user
        self.password = password
        self.ipaddr = ipaddr
        self.port = port
        self.headers = {
            "Accept": "application/json",
            "Content-type": "application/json",
            "X-EMC-REST-CLIENT": "true"
        }
        self.session = requests.Session()

    def get(self, url):
        """Send HTTP GET request and return raw data of the response."""
        return self.session.get(url,
                                headers=self.headers,
                                auth=(self.user, self.password),
                                verify=False)

    def post(self, url, data):
        """Send HTTP POST request and return raw data of the response."""
        return self.session.request("POST",
                                    url,
                                    headers=self.headers,
                                    data=json.dumps(data),
                                    verify=False)

    def get_resource_attributes(self, resource):
        """Return all attributes of the resource."""
        url = "https://{}:{}/api/types/{}?compact=True&fields=" \
            "attributes.name".format(self.ipaddr, self.port, resource)
        r = self.get(url)
        j = json.loads(r.content)
        attrs = [x["name"] for x in j["content"]["attributes"]]
        return ",".join(attrs)

    def get_resource_instances(self, resource):
        """Collection query, return json decoded object."""
        attrs = self.get_resource_attributes(resource)
        url = "https://{}:{}/api/types/{}/instances?compact=True&fields" \
            "={}".format(ipaddr, port, resource, attrs)
        r = self.get(url)
        return json.loads(r.content)

    def get_resource_instance_by_name(self, resource, name):
        """Instance query, return json decoded object."""
        attrs = self.get_resource_attributes(resource)
        url = "https://{}:{}/api/instances/{}/name:{}?compact=True&fields" \
            "={}".format(ipaddr, port, resource, name, attrs)
        r = self.get(url)
        return json.loads(r.content)

    def update_csrf_token(self):
        """Update CSRF token to send POST/DELETE requests."""
        if "EMC-CSRF-TOKEN" not in self.headers:
            url = "https://{}:{}/api/types/user/instances".format(self.ipaddr,
                                                                  self.port)
            r = self.get(url)
            self.headers["EMC-CSRF-TOKEN"] = r.headers["EMC-CSRF-TOKEN"]

    def create_resource_instance(self, url, data):
        """Create resource instance, return json decoded object.."""
        self.update_csrf_token()
        r = self.post(url, data)
        return json.loads(r.content)

    def get_disk_groups(self):
        """Get disk groups."""
        return self.get_resource_instances("diskGroup")

    def create_pool(self, name):
        """Create pool."""
        resource = "pool"

        # Check if pool already exists.
        pool = self.get_resource_instance_by_name(resource, name)

        if "content" in pool:
            print("Pool {} already exists.".format(name))
            return pool["content"]["id"]

        url = "https://{}:{}/api/types/{}/instances".format(self.ipaddr,
                                                             self.port,
                                                             resource)

        # Create a pool with Capacity Tier and Raid 1/0.
        tier_type = 30  # Capacity tier
        num_disks = 2  # Number of disks
        raid_type = 7  # Raid 1/0
        stripe_width = 2  # A 2 disk group, for Raid 1/0 1+1 configuration
        pool_type = 1  # Traditional pool

        # Select suitable disk group.
        disk_groups = self.get_disk_groups()
        disk_group_id = [x["content"]["id"] for x in disk_groups["entries"]
                         if x["content"]["tierType"] == tier_type
                         and x["content"]["unconfiguredDisks"] >= num_disks][0]
        print("Select disk group: {}.".format(disk_group_id))

        data = {
            "name": name,
            "addRaidGroupParameters": [
                {
                    "dskGroup": {
                        "id": disk_group_id
                    },
                    "numDisks": num_disks,
                    "raidType": raid_type,
                    "stripeWidth": stripe_width
                }
            ],
            "type": pool_type,
            "isFASTCacheEnabled": False
        }

        print("Create new pool: {}.".format(name))
        return self.create_resource_instance(url, data)["content"]["id"]
```

```
def create_lun(self, name, size, pool_id):
    """Create Lun."""
    resource = "storageResource"

    # Check if lun already exists.
    lun = self.get_resource_instance_by_name(resource, name)

    if "content" in lun:
        print("Lun {} already exists.".format(name))
        return lun["content"]["id"]

    url = "https://{}:{}/api/types/{}/action/createLun".format(ipaddr,
                                                      port,
                                                      resource)

    data = {
        "name": name,
        "lunParameters": {
            "pool": {
                "id": pool_id
            },
            "size": size
        }
    }

    print("Create new lun: {}.".format(name))
    return self.create_resource_instance(url, data)["content"][resource][
        "id"]

def create_host(self, name):
    """Create Host."""
    resource = "host"

    # Check if host already exists.
    host = self.get_resource_instance_by_name(resource, name)

    if "content" in host:
        print("Host {} already exists.".format(name))
        return host["content"]["id"]

    print("Create new host: {}.".format(name))
    url = "https://{}:{}/api/types/{}/instances".format(self.ipaddr,
                                                 self.port,
                                                 resource)

    data = {
        "name": name,
        "type": 1
    }

    return self.create_resource_instance(url, data)["content"]["id"]

def create_host_initiator(self, initiator_iqn, host_id):
    """Create Host Initiator."""
    resource = "hostInitiator"

    # Check if host already exists.
    url = 'https://{}:{}/api/types/{}/instances?compact=True' \
          '&filter=initiatorId eq "{}"&fields=id'.format(ipaddr, port,
                                                          resource,
                                                          initiator_iqn)
    r = self.get(url)
    j = json.loads(r.content)

    if len(j["entries"]) != 0:
        host_initiator = j["entries"][0]
        if "content" in host_initiator:
            print(
                "Host initiator {} already exists.".format(initiator_iqn))
            return host_initiator["content"]["id"]

    # Create new hostInitiator.
    url = "https://{}:{}/api/types/{}/instances".format(self.ipaddr,
                                                         self.port,
                                                         resource)

    data = {
        "host": {"id": host_id},
        "initiatorType": 2,  # iSCSI initiator
        "initiatorWWNorIqn": initiator_iqn
    }

    print("Create new host initiator: {}.".format(initiator_iqn))
    return self.create_resource_instance(url, data)["content"]["id"]

def attach_lun_to_host(self, lun_id, host_id):
    """Attach Lun to Host."""
    url = "https://{}:{}/api/instances/storageResource/{}/action/" \
          "modifyLun?compact=true&visibility=Engineering&timeout=5" \
          "".format(self.ipaddr, self.port, lun_id)

    data = {
        "lunParameters": {
            "hostAccess": [
                {
                    "accessMask": 4,
                    "host": {
                        "id": host_id
                    }
                }
            ]
        }
    }

    print("Attach Lun {} to Host {}.".format(lun_id, host_id))
    self.update_csrf_token()
    self.post(url, data)
```

```
if __name__ == "__main__":
    user = "admin"
    password = "Password123!"
    ipaddr = "10.229.114.32"
    port = 443

    client = UnityRestClient(user, password, ipaddr, port)

    # Create a pool.
    pool_id = client.create_pool("test_pool_1")

    # Create a lun.
    size = 10 * 1024 * 1024 * 1024  # 10GB
    lun_id = client.create_lun("test_lun_1", size, pool_id)

    # Create a host.
    host_id = client.create_host("test_host_1")

    # Create a host initiator.
    initiator_iqn = "iqn.1994-05.com.redhat:2bfbc0884dc4"
    client.create_host_initiator(initiator_iqn, host_id)

    # Attach lun to host, so user could access the lun in the server.
    client.attach_lun_to_host(lun_id, host_id)
```